SISTEMI OPERATIVI

Danilo Vizzarro
info@danilovizzarro.it

8 Maggio 2006

Indice

Capitolo 1

Architetture dei Sistemi Operativi

1.1 Dalle Strutture Monolitiche al Microkernel

Un sistema operativo può essere caratterizzato da diversi tipi di architetture. La struttura più semplice, si basa su un blocco monolitico avente lo scopo di fornire la massima funzionalità utilizzando il minimo spazio. Esso racchiude tutte le funzioni del sistema operativo, permettendo un'elevata velocità di accesso alle procedure, a discapito della difficoltà che si riscontrerebbe qualora si voglia modificare il sistema operativo. Un esempio di sistema monolitico è l'MS-DOS.
Per rendere più versatile il sistema, sono state progettate delle architetture più complesse. Ne è un esempio l'architettura stratificata che prevede che il sistema sia suddiviso in strati, da quello fisico (0), a quello a interfaccia utente (n) e che ogni strato sia composto da strutture dati e procedure che possono essere chiamate solo dagli strati superiori. Il principale vantaggio di tale architettura è la modularità: gli strati sono composti in modo tale che ciascuno di essi utilizzi solo funzioni e servizi che appartengono agli strati inferiori e consideri solo le azioni che esse compiono senza entrare nel merito di come queste siano realizzate. La stratificazione semplifica la progettazione e la realizzazione di un sistema, tuttavia, tale architettura, presenta alcuni svantaggi riscontrabili nella difficoltà di definire in modo accurato gli strati, dovuta alla possibilità di utilizzare solo strati che si trovano ad un livello inferiore e nella minore efficienza della struttura stratificata rispetto alle altre architetture, dovuta alla necessità di effettuare chiamate tra i vari livelli. Una architettura di questo tipo può essere ottimizzata implementando meno strati con più funzioni. Molte versioni dei sistemi Unix si basano su sistemi stratificati.
Un'altra architettura prevede l'utilizzo di un microkernel dal quale devono

essere rimossi tutti i componenti non essenziali al funzionamento del sistema, che sarebbero poi realizzati come servizi a livello utente. Il microkernel offre servizi di gestione dei processi, della memoria e della comunicazione attraverso scambi di messaggi e presenta numerosi vantaggi tra cui: la maggiore sicurezza dovuta alle dimensioni limitate del microkernel che permette di eseguire più facilmente operazioni di testing e correzioni, l'adattabilità del sistema operativo alle diverse architetture, la facilità di estensione e di modifica del sistema. Un esempio di sistema basato su questo tipo di architettura è il MacOS X. I sistemi Windows NT utilizzano invece una struttura ibrida, a strati e a microkernel.

Capitolo 2

Il Processore

2.1 Virtualizzazione del Processore

Con la multiprogrammazione si tengono contemporaneamente in memoria più processi, e quando uno di questi deve attendere un evento, il sistema operativo gli sottrae il controllo della cpu per cederlo ad un altro processo. A tal fine si utilizza lo scheduling della cpu.
Ogni processo segue un ciclo di elaborazione: dopo essere stato creato, non appena è pronto per controllare la cpu, comincia l'esecuzione con una sequenza di operazioni d'elaborazione (cpu burst), seguita da una sequenza di operazioni di I/O (I/O burst), quindi da un'altra sequenza di operazioni della cpu, di nuovo una sequenza di operazioni di I/O e così via. L'ultima sequenza di operazioni della cpu si conclude con una richiesta al sistema di terminare l'esecuzione.
Un programma in cui prevalgono operazioni di I/O viene detto I/O bound e generalmente produce molte sequenze di operazioni di cpu di breve durata, mentre un programma in cui prevalgono le operazioni d'elaborazione, viene detto cpu bound e produce poche sequenze di operazioni molto lunghe.
Lo scheduler si occupa di scegliere quale, tra i processi pronti, assegnare alla cpu, in particolare interviene in 4 diverse circostanze:

1. Se un processo passa dallo stato di esecuzione allo stato di attesa (es. I/O);

2. Se un processo passa dallo stato di esecuzione allo stato ready (es. interrupt);

3. Se un processo passa dallo stato di attesa allo stato ready (es. completamento dell'I/O);

4. Se un processo termina.

Colui che invece si occupa di passare effettivamente il controllo della cpu da un processo ad un altro è il dispatcher che interviene secondo le politiche di scheduling quando:

- si deve cambiare contesto;
- si passa al modo utente;
- si salta alla giusta posizione del programma utente per riavviarne l'esecuzione.

2.2 Algoritmi di Scheduling della CPU

Gli algoritmi di scheduling vengono realizzati secondo alcuni criteri finalizzati alla massimizzazione dell'utilizzo della cpu e all'ottimizzazione del tempo di computazione tra cui:

- l'utilizzo della cpu, che deve essere attiva per il maggior tempo possibile;
- il troughput che identifica il numero di processi che completano la loro esecuzione nell'unità di tempo ed è utilizzato per minimizzare il tempo di attesa di ogni processo prima di ottenere l'uso del processore;
- il tempo di completamento, ovvero, il tempo totale che un processo spende da quando viene sottoposto al processore;
- il tempo d'attesa, ovvero, la somma degli intervalli d'attesa passati nella coda dei processi pronti;
- il tempo di risposta, ovvero, il tempo che intercorre tra la sottomissione della richiesta e la prima risposta prodotta.

Non esiste un algoritmo ideale in tutte le circostanze ma è opportuno valutare volta per volta quale algoritmo utilizzare.

L'algoritmo FCFS (First Come First Served) è il più semplice algoritmo basato su una coda FIFO, che prevede che il primo processo ad essere inserito nella coda d'attesa del processore, sia anche il primo processo ad essere servito dallo stesso. Il tempo di attesa varia a seconda del tempo impiegato dalla cpu, per eseguire la computazione delle operazioni di ogni processo precedente. Se all'inizio della coda sono presenti processi con una lunga sequenza di operazioni da eseguire e alla fine della coda sono presenti processi

più brevi, il tempo d'attesa di questi ultimi sarà molto elevato. Se invece si presenta la situazione opposta il tempo d'attesa medio calerà notevolmente. L'algoritmo SJF (Short Job First) assegna la cpu al processo con la successiva sequenza di operazioni della cpu (cpu burst) più breve. A parità di lunghezza si applica lo scheduling FCFS. L'algoritmo SJF può essere sviluppato sia in modo preemptive (con prelazione) che nonpreemptive (senza prelazione). Nell'SJF preemptive, se nella coda dei processi pronti si presenta un processo con un CPU burst più breve del processo attualmente in esecuzione, si interrompe il processo in esecuzione e si cede la CPU al nuovo arrivato. Nell'SJF nonpreemptive è permesso al processo in esecuzione di terminare la propria sequenza di operazioni della cpu, prima di dare il controllo della cpu al nuovo processo arrivato.
L'algoritmo per priorità, tiene conto delle priorità associate a ciascun processo e prevede che la cpu sia assegnata al processo con priorità più alta. Anche lo scheduling con priorità può essere di tipo preemptive o non preemptive. Nel primo caso se si presenta un processo con priorità più alta di quello in esecuzione, si interrompe quest'utimo e si cede la cpu al nuovo processo arrivato. Nel caso non preemptive, ci si limita a porre il processo arrivato in testa alla coda dei processi pronti. Un problema che si può verificare utilizzando questo algoritmo è la starvation, ovvero l'attesa infinita cui si può andare incontro se vengono messi in attesa dei processi con bassa priorità e continuino ad arrivare processi con priorità maggiore.
L'algoritmo round-robin è stato progettato per i sistemi a partizione di tempo e si basa su una coda circolare. Risulta simile all'algoritmo FCFS ed ha capacità di prelazione. Ogni nuovo processo viene inserito alla fine della coda FIFO e per la computazione dei processi viene assegnata a ciascun processo una quantità di tempo fissata detta quanto di tempo. Lo scheduler percorre la coda assegnando la cpu a ciascun processo per la durata massima del quanto di tempo, al termine del quale possono verificarsi due casi:

- se la cpu termina l'elaborazione del processo prima dello scadere del quanto si restituisce il controllo alla cpu;
- altrimenti:
    1. Un temporizzatore invia un segnale d'interruzione al sistema operativo;
    2. Si esegue un cambio di contesto;
    3. Si aggiunge il processo alla fine della coda dei processi pronti;
    4. Tramite lo scheduler si seleziona un nuovo processo.

C'è da considerare che i cambi di contesto sprecano tempo di cpu e pertanto la grandezza del quanto di tempo va calibrato in modo opportuno.

L'algoritmo a code multiple prevede che i processi vengano divisi in due code: quelli da eseguire in foreground con alta interattività con l'utente e quelli da eseguire in background. Ciascuna coda viene eseguita con un proprio algoritmo.
L'algoritmo a code multiple con retroazione permette ai processi di migrare da una coda all'altra, ossia, un processo che utilizza molto tempo di elaborazione della cpu, può spostarsi in una coda con priorità più bassa e viceversa; analogamente si può evitare l'attesa indefinita di un processo spostandolo in una coda con priorità più alta. I parametri di questo algoritmo sono:

1. Il numero di code;
2. L'algoritmo di scheduling di ciascuna coda;
3. Il metodo usato per determinare quando spostare un processo in una coda con priorità maggiore o minore;
4. Il metodo usato per determinare in quale coda si deve mettere un processo quando viene richiesto un servizio.

Se il sistema presenta più unità di elaborazione, lo scheduling può essere eseguito in 2 modi:

1. Ogni unità di elaborazione esamina la coda dei processi pronti e ne esegue uno;
2. Si seleziona una cpu che si occupa dello scheduling di tutte le altre. Tale cpu prende il nome di master server mentre lo schema viene detto di multielaborazione asimmetrica.

Nei sistemi real-time bisogna garantire il completamento delle funzioni critiche entro un tempo definito eseguendo generalmente un processo alla volta. I processi si presentano insieme ad una dichiarazione di tempo entro il quale devono essere completati; solo se la cpu ritiene che sia possibile garantire il rispetto di questo tempo, questi vengono accettati.

Capitolo 3

I Processi

3.1 Comunicazione tra Processi

Le funzioni IPC (Inter Process Comunication) permettono ai processi di comunicare tra loro attraverso lo stesso canale di comunicazione e di sincronizzare le proprie azioni, anche senza condividere lo spazio di indirizzi. Le tecniche di IPC si basano sullo scambio di messaggi che si realizza utilizzando due funzioni primitive (send e recive), utilizzate rispettivamente per inviare e ricevere messaggi, e sull'utilizzo di due tipi diversi di comunicazione: diretta o indiretta.
Con la comunicazione diretta, ogni processo deve nominare esplicitamente l'altro interlocutore, sia esso il ricevente o il mittente della comunicazione. Tramite la funzione send(P,messaggio) è possibile inviare un messaggio al processo P, mentre per mezzo della funzione recive(Q,messaggio) è possibile ricevere un messaggio dal processo Q. Nella comunicazione diretta, il canale di comunicazione è associato esclusivamente ai 2 processi e tra ciascuna coppia di processi esiste un solo canale di comunicazione.
Con la comunicazione indiretta, i messaggi si inviano a delle porte dette mailbox. Per mezzo della funzione send(A,messaggio) è possibile inviare un messaggio alla mailbox A, mentre con la funzione recive(A,messaggio) è possibile ricevere un messaggio dalla mailbox A. Ciascuna mailbox è identificata in modo univoco e può essere associata a due o più processi. Nel caso in cui una mailbox sia condivisa da 3 processi e accada che p1 invia un messaggio ad A mentre p2 e p3 eseguono un recive da A, nasce il problema di sapere quale processo riceverà il messaggio. La soluzione varia a seconda dello schema che si sceglie. Si può imporre che un canale sia associato al massimo a due processi, o in alternativa imporre che un solo processo per volta possa eseguire l'operazione di recive, o consentire al sistema di decidere arbitrariamente a

quale processo arriverà il messaggio. Nella comunicazione indiretta, il canale di comunicazione viene quindi stabilito solo se si condivide la stessa mailbox e può essere associato a più processi, inoltre, ad ogni coppia di processi comunicanti, possono essere associati più canali di comunicazione diversi. Il sistema operativo permette ai processi di creare una nuova mailbox, di inviare e ricevere messaggi tramite la mailbox, nonché di rimuoverla al termine della comunicazione. La mailbox farà parte dello spazio di indirizzi del processo e verrà eliminata al termine dell'esecuzione del processo. L'invio di messaggi può essere sincrono se il processo mittente si blocca in attesa che il processo destinatario o la mailbox di destinazione abbiano ricevuto il messaggio, oppure asincrono se il processo invia il messaggio e riprende la propria esecuzione. Si parla inoltre di ricezione sincrona se il ricevente si blocca in attesa dell'arrivo di un messaggio, e di ricezione asincrona se il ricevente continua la sua esecuzione finchè non riceve un messaggio valido oppure un valore nullo. A ogni mailbox è associata una coda in cui risiedono i messaggi d'attesa che possono essere prelevati dal ricevente. La capacità delle code può essere:

- uguale a zero se il canale non può avere messaggi in attesa al suo interno;
- limitata se al suo interno possono risiedere al massimo n messaggi;
- illimitata se la coda ha una lunghezza virtualmente infinita.

3.2 Sincronizzazione dei Processi

I processi cooperanti possono influenzare o essere influenzati da altri processi in esecuzione che condividono uno spazio logico di indirizzi oppure dati. Per evitare che due o più processi leggano o scrivano su un dato condiviso e che il risultato finale dipenda dall'ordine con cui vengono eseguiti i processi, occorre sincronizzare gli stessi assicurando che un solo processo alla volta possa modificare i dati condivisi. Si definiscono sezioni critiche, quei segmenti di codice in cui il processo può modificare variabili comuni e altri dati condivisi. Ogni processo deve essere autorizzato a entrare nella propria sezione critica. La sezione di codice che realizza questa richiesta è detta sezione d'ingresso, la sezione che segue quella critica è detta sezione d'uscita, mentre la restante parte di codice è detta sezione non critica. Per risolvere il problema illustrato precedentemente i processi devono soddisfare i seguenti requisiti:

1. Mutua esclusione: se un processo è in esecuzione nella sua sezione critica, nessun altro processo potrà essere in esecuzione nella propria sezione critica;

2. Progresso: se nessun processo è in esecuzione nella sua sezione critica e qualche processo desidera entrare nella propria sezione critica, la scelta del processo che può accedere per primo alla propria sezione critica spetta solo ai processi che si trovano nelle rispettive sezioni d'ingresso;

3. Attesa illimitata: se un processo ha già richiesto l'ingresso nella sua sezione critica, c'è un limite massimo di volte che si consente ad altri processi di entrare nelle proprie sezioni critiche prima che si accordi la richiesta al primo processo.

Il problema della sezione critica può essere risolto con diversi algoritmi. Si considerino due processi P0 e P1.
Algoritmo 1. I due processi condividono una variabile 'turno', che può assumere valore 0 o 1. A seconda di questo valore, il processo con indice uguale a turno può accedere alla sua sezione critica, dopo di che il valore di turno cambia. Questa soluzione non soddisfa il requisito di progresso.
Algoritmo 2. Viene condiviso un vettore booleano 'pronto' di dimensione pari 2, inizializzandone i valori a false. Quando un processo è pronto ad accedere alla propria sezione critica, questo imposta a true il valore corrispondente nel vettore 'pronto' e controlla che l'altro processo non abbia il valore corrispondente nel vettore impostato a true, in tal caso attende che l'altro processo termini, altrimenti accede alla sua sezione critica e, al termine, reimposta il valore false nel vettore 'pronto'.
Algoritmo 3. Si hanno 2 strutture: un vettore booleano 'pronto' di dimensione pari 2 e una variabile condivisa 'turno'. Quando un processo vuole entrare nella propria sezione critica, imposta a true il bit corrispondente nel vettore pronto, assegna a 'turno' l'indice dell'altro processo per permettergli di accedere alla sezione critica; se anche il valore del vettore 'pronto' corrispondente all'altro processo è impostato a true e il turno è dell'altro processo e per questo rimane in attesa che non valga più almeno una di queste condizioni, altrimenti entra nella sezione critica e quando ha terminato le operazioni, imposta a false il bit corrispondente al processo corrente nel vettore 'pronto'.
Nel caso in cui la sincronizzazione riguardi più di due processi, si utilizzano gli algoritmi di seguito.
Algoritmo del fornaio. Ha lo scopo di risolvere la sezione critica per n processi. Quando un processo deve entrare nella sua sezione critica, lo segnala in un vettore booleano; gli viene attribuito un numero progressivo e viene riportato a false il valore nel vettore. Per decidere quale processo debba entrare per primo nella propria sezione critica, si confrontano i numeri progressivi di quei processi che hanno già acquisito tale numero: avrà accesso alla propria

sezione critica, il processo con numero progressivo minore e, a parità di numero, quello con l'indice inferiore. Svolte le proprie operazioni nella sezione critica, il processo reimposta il suo numero progressivo a 0.
Il semaforo è invece uno strumento di sincronizzazione costituito da una variabile intera da inizializzare a un valore non negativo, cui si può accedere per mezzo di 2 operazioni che, ancora una volta, devono essere eseguite in modo atomico:

- Wait che decrementa il valore della variabile di una unità e induce un'attesa sul processo che la esegue in caso il valore del semaforo diventi negativo;
- Signal che incrementa il valore della variabile di un'unità: se il valore ottenuto non è positivo, uno dei processi precedentemente messo in attesa da un'operazione di wait, viene liberato dall'attesa.

Il vantaggio dei semafori di questo tipo consiste nel non dover compiere alcun cambio di contesto quando un processo attende l'autorizzazione all'accesso.

Sono stati poi introdotti altri due costrutti di sincronizzazione quale la regione critica e il monitor.
La regione critica non elimina tutti gli errori di sincronizzazione ma ne riduce il numero. Prevede che si dichiari la variabile che deve essere condivisa e che si possa accedere a tale variabile solo dall'interno di un'istruzione region. Tale istruzione contiene un'espressione booleana: se questa risulta vera, si esegue l'istruzione, altrimenti si ritarda l'esecuzione finchè non risulta vera.
Il monitor funziona in questo modo: vengono definite delle variabili condizionali (x,y) sulle quali è possibile eseguire solo 2 operazioni:

- x.wait che sospende il processo che effettua questa operazione finchè un altro processo non effettua una x.signal;
- x.signal che risveglia un processo; se nessun processo è in attesa tale operazione non ha alcun effetto.

3.3 Il Deadlock

Se un processo è in attesa di risorse trattenute da altri processi anch'essi in stato di attesa, può verificarsi una situazione di stallo detta deadlock.
Un sistema è composto da un numero finito di risorse da distribuire tra più processi in competizione, rispettando la seguente sequenza:

1. Richiesta: se un processo richiede una risorsa non immediatamente disponibile, deve attendere finchè può acquisirla;

2. Uso: il processo utilizza la risorsa;

3. Rilascio: finito l'utilizzo, il processo rilascerà la risorsa.

Ci sono 4 condizioni che devono verificarsi contemporaneamente affinché si possa parlare di deadlock:

- mutua esclusione: almeno una risorsa deve poter essere utilizzata da un processo per volta;
- possesso e attesa: i processi che detengono risorse assegnategli in precedenza possono richiedere nuove risorse e attendere che queste si liberino se sono occupate;
- impossibilità di prelazione: le risorse precedentemente assegnate non possono essere forzatamente revocate a un processo ma devono essere esplicitamente rilasciate dal processo che le detiene;
- attesa circolare: ci deve essere una lista circolare di processi ciascuno dei quali è in attesa di una risorsa posseduta dal processo seguente nella lista.

Per gestire le situazioni di deadlock, vengono utilizzate 4 strategie.
Prevenire il deadlock: si cerca di evitare il verificarsi contemporaneo delle 4 condizioni su citate. Tuttavia non è possibile negare la mutua esclusione in quanto se una risorsa non è condivisibile lo è intrinsecamente. Il 'possesso e attesa' si evita facendo in modo che tutti i processi richiedano tutte le risorse di cui hanno bisogno prima di iniziare l'esecuzione o facendo in modo che un processo possa richiedere una risorsa solo se non ne possiede altre. L'impossibilità di prelazione si può evitare facendo in modo che, se un processo che possiede una o più risorse ne richiede un'altra non immediatamente disponibile, si eserciti la prelazione su tutte le risorse in suo possesso. L'attesa circolare può essere eliminata in vari modi per esempio numerando univocamente tutte le risorse. I processi potranno richiedere risorse quando lo desiderano, ma tutte le richieste devono essere effettuate in ordine numerico. Con questa regola, il grafo di allocazione delle risorse non potrà mai presentare dei cicli.
Evitare il deadlock: un sistema si trova in uno stato sicuro, se è in grado di assegnare risorse a ciascun processo in un certo ordine, impedendo situazioni di deadlock. Questo può essere evitato per mezzo di 2 algoritmi. L'algoritmo con grafo di assegnazione delle risorse, prevede che se le risorse di un sistema hanno una sola istanza, allora per garantire lo stato sicuro si aggiunge al grafo di assegnazione delle risorse un altro tipo di arco, detto arco di reclamo

(P_i->R_j) che indica la possibilità che P_i richieda R_j. Le risorse devono essere reclamate a priori nel sistema. Se P_i richiede R_j, l'arco di reclamo può divenire arco di richiesta se e solo se non si crea un ciclo. L'algoritmo del banchiere prevede che quando una risorsa può essere assegnata a un processo, si simuli l'assegnazione delle risorse al processo, aggiornando i valori relativi alle risorse disponibili, a quelle assegnate e a quelle necessarie. Si controlla quindi in maniera iterativa, che lo stato di assegnazione delle risorse sia sicuro. Se alla fine tutti i processi saranno portati a termine, allora lo stato di assegnazione delle risorse resterà sicuro e la risorsa richiesta verrà assegnata al processo.
Rilevare il problema e ripristinare il sistema: questa tecnica consiste nell'individuare il deadlock e risolverlo. Ci si limita a sorvegliare le richieste e i rilasci delle risorse. Ogni volta che una risorsa viene richiesta o rilasciata si aggiorna il grafo delle risorse controllando che non si sia verificato un ciclo. Se esiste un ciclo uno dei processi viene terminato e se questo non elimina il deadlock ne viene terminato un altro.
Ignorare il problema: l'approccio più semplice tuttavia, consiste nell'ignorare del tutto il problema. Il deadlock accade molto raramente mentre i crash di sistema sono molto frequenti. Per questa ragione si può decidere di non pagare in termini di prestazioni al fine di evitare il deadlock.

3.4 La Starvation

Per starvation si intende l'attesa infinita cui può andare incontro utilizzando alcuni algoritmi di scheduling. Ne è un esempio lo scheduling per priorità (individuato nell'ambito degli algoritmi di scheduling della cpu) che tiene conto delle priorità associate a ciascun processo e prevede che la cpu sia assegnata al processo con priorità più alta.
La starvation si verifica se vengono messi in attesa dei processi con bassa priorità e continuino ad arrivare processi con priorità maggiore. La soluzione a questo tipo di problema è l'aging che consiste nell'aumentare gradualmente le priorità dei processi che attendono da più tempo di essere eseguiti.
La starvation si verifica anche con lo scheduling per brevità SSTF (individuato nell'ambito degli algoritmi di scheduling della del disco) che prevede che venga soddisfatta la ricerca che da il minimo seek time rispetto alla posizione corrente della testina.

Capitolo 4

Gestione della Memoria

4.1 La Tecnica di Paginazione

La tecnica di paginazione prevede la divisione della memoria fisica, in blocchi di memoria detti pagine fisiche e la divisione della memoria logica in pagine logiche, dette semplicemente pagine. Anche la memoria ausiliaria è divisa in pagine della stessa dimensione di quelle della memoria fisica. Ogni indirizzo generato dalla cpu viene diviso in due parti: un numero di pagina 'p' e uno scostamento di pagina 'd'. Il numero di pagina è utilizzato come indice per la tabella delle pagine che contiene l'indirizzo di base della memoria fisica, ovvero il primo indirizzo della pagina. Ogni processo è dotato di una propria tabella delle pagine che realizza la corrispondenza fra pagine logiche e pagine fisiche effettivamente assegnate al processo. Il sistema operativo, memorizza nella 'tabella dei blocchi di memoria' se le pagine sono libere o assegnate ai processi. Se la dimensione del processo è indipendente dalla dimensione della pagina, può divenire rilevante il problema della frammentazione interna, dovuta allo spazio non occupato nell'ultima partizione assegnata a un processo. La frammentazione interna può essere ridotta riducendo la dimensione delle pagine, ma ciò comporterebbe un maggior carico di lavoro dovuto alla maggiore dimensione della tabella delle pagine.
Quando si deve eseguire un processo, si controlla che ci siano abbastanza blocchi liberi, si assegnano tali blocchi al processo e si carica la prima pagina, inserendo il numero del blocco di memoria nella tabella delle pagine relativa al processo in questione. Al programma utente sembrerà di avere a disposizione uno spazio di memoria contiguo, mentre in realtà questo è sparso nella memoria fisica.
Ogni processo contiene nel PCB (Process Control Block) un collegamento alla propria tabella delle pagine e i sistemi operativi utilizzano dei propri

metodi per memorizzare la stessa tabella. Uno dei metodi più veloci è la realizzazione della tabella tramite registri. Questa però non è concretamente applicabile a causa della grande dimensione della tabella delle pagine e dell'elevato costo dei registri. Si utilizza pertanto un apposito registro PTBR (Page Table Base Register) che punta alla tabella. Il cambio di contesto viene così velocizzato, perché è sufficiente aggiornare questo registro ogni volta che viene cambiato il processo in esecuzione sul processore. Il problema di questa tecnica è che per accedere a un determinato indirizzo in memoria occorre prima accedere alla tabella delle pagine, anch'essa posizionata nella RAM, raddoppiando di fatto il numero di accessi alla memoria.
Per la protezione dello spazio di memoria dei processi e del sistema operativo, si memorizzano nella tabella delle pagine dei bit di protezione che determinano se una pagina può essere solo letta o anche scritta. Se si tenta di scrivere su una pagina che può essere solo letta, viene generata un'eccezione al sistema operativo. Solitamente si associa ad ogni pagina un bit di validità che indica se una pagina logica associata ad un processo è utilizzata o meno.
La tabella delle pagine può assumere grandi dimensioni, pertanto è opportuno trovare un sistema per velocizzare la ricerca delle pagine.
Una tecnica è la paginazione gerarchica, che consiste nello spezzare la tabella delle pagine in 2 tabelle, una di primo livello contenente i collegamenti alla tabella di secondo livello, contenente a sua volta gli indirizzi di base delle pagine fisiche. Il vantaggio è la possibilità di tenere in memoria solo le tabelle di secondo livello effettivamente necessarie riducendo l'occupazione di memoria.
Nelle architetture a 64 bit si utilizza invece una funzione di tipo hash che prende come argomento il numero di pagina logica e restituisce l'indice della tabella anch'essa di tipo hash. Questa tecnica risulta utile in presenza di indirizzi sparsi.
Con la tecnica della tabella delle pagine invertita si ha un'unica tabella mantenuta dal sistema operativo e non una tabella per processo come avveniva per le tecniche di paginazione precedenti. La tabella delle pagine invertita ha un elemento per ogni pagina di memoria fisica in cui è memorizzato il 'pid' del processo che possiede quella pagina e il suo indirizzo logico. Per accedere alla pagina, occorre scorrere la tabella cercando la corrispondente pid-pagina cercata. Questo comporta la riduzione della memoria utilizzata per le tabelle a discapito dell'aumento di tempo di ricerca di una pagina.
L'ultimo vantaggio della paginazione, è la possibilità di condividere del codice comune tra i vari processi ponendolo in pagine fisiche che possono essere accedute da diversi processi, a patto che il codice condiviso sia di tipo rientrante, ovvero, non si automodifichi.

4.2 La Tecnica di Segmentazione

La segmentazione è un altro schema di gestione della memoria. Lo spazio di indirizzi logici viene diviso in segmenti, ciascuno dei quali ha un nome e una lunghezza. Gli indirizzi specificano sia il nome sia lo scostamento all'interno di un segmento, quindi ogni indirizzo logico è dato dalla coppia <idsegmento, sostamento>. Esempi di segmenti possono essere le variabili locali, la pila per le chiamate di procedure, il codice di ogni funzione e le variabili locali di ciascuna funzione.
Per poter implementare la segmentazione, occorre poter tradurre gli indirizzi formati dalla coppia <idsegmento, sostamento> negli indirizzi unidimensionali della memoria fisica. A tal fine viene utilizzata la tabella dei segmenti contenente un elemento per ogni segmento a sua volta diviso in 'base' contenente l'indirizzo fisico iniziale della memoria del segmento e 'limite' contenente la lunghezza dello stesso. L'indirizzo logico è diviso in due parti: un numero di segmento 's' e uno scostamento in tale segmento 'd'. Il numero di segmento si utilizza come indice della tabella dei segmenti, mentre lo scostamento 'd' deve essere compreso tra 0 e il limite del segmento, altrimenti viene inviato un segnale di eccezione al sistema operativo. Se tale condizione è rispettata, si somma lo scostamento alla base del segmento per produrre l'indirizzo di memoria fisica vero e proprio.
La protezione della memoria è assicurata aggiungendo un bit di protezione alla tabella dei segmenti. Un altro vantaggio della segmentazione è la condivisione di codici o dati. Si può condividere qualsiasi informazione definita come segmento, ed è possibile anche condividere un intero programma composto da diversi segmenti, oppure una parte di esso.
Con la segmentazione non si ha il problema della frammentazione interna in quanto i segmenti hanno dimensione variabile. Resta però il problema della frammentazione esterna dovuta al caricamento dei processi in memoria, in spazi disponibili sempre più piccoli.

4.3 La Memoria Virtuale

Con la tecnica della memoria virtuale, è possibile evitare di caricare in memoria centrale gli interi processi e fare in modo che i programmi possano disporre di più memoria di quella fisica effettivamente disponibile, mostrando sia la memoria fisica, che quella logica, come un unico spazio di indirizzi. Tuttavia le istruzioni da eseguire devono trovarsi all'interno della memoria fisica e per far fronte a tale requisito, si colloca l'intero spazio di indirizzi logici del processo, nella memoria fisica. La sovrapposizione delle sezioni è nota come

overlay.

L'utilizzo della memoria virtuale è vantaggioso per diverse ragioni:

- un programma non è più vincolato ad avere una dimensione massima pari alla memoria fisica disponibile;
- ogni utente impiega meno memoria fisica rendendo possibile l'esecuzione di più programmi contemporaneamente in quanto non vengono caricate in memoria tutte le pagine ma solo quelle necessarie;
- per caricare ogni programma nella memoria sono necessarie meno operazioni di I/O.

L'esecuzione di un processo avviene tramite un sistema di paginazione su richiesta dove i processi risiedono in memoria secondaria e vengono caricati in memoria centrale per essere eseguiti. Il caricamento avviene secondo il criterio del lazy swapping che prevede che non venga mai caricato in memoria l'intero processo ma solo le pagine effettivamente necessarie. E' quindi necessario che l'architettura fornisca una tabella delle pagine che permetta di rilevare quali pagine sono presenti in memoria e quali no. Questo si realizza contrassegnando un elemento come non valido attraverso il bit di validità e attraverso una memoria ausiliaria che conserva le pagine non caricate in memoria. La sezione del disco che si occupa di ciò è l'area di avvicendamento (swap space). Quando viene richiesto un dato non presente in memoria, viene generata un'eccezione di pagina mancante detta page fault trap. Di conseguenza si individua un blocco di memoria libero e viene inviata la richiesta di trasferimento della pagina al controller del disco. Il procedimento termina aggiornando a valido il bit della pagina trasferita. Il modulo del sistema operativo che si occupa della sostituzione delle pagine è il paginatore. Salvando lo stato del processo (ovvero i registri, i codici di condizione e i contatore di programma) che è stato interrotto da un page fault trap, si può riavviare il processo esattamente nello stesso punto e nello stesso stato nel quale si trovava quando era stato interrotto. Questo permette di eseguire un processo anche se alcune sue pagine non sono in memoria.
Il tempo di accesso effettivo alla memoria, è direttamente proporzionale alla frequenza delle assenze di pagine, per questo è importante tener bassa la frequenza di quest'ultime.
Per migliorare le prestazioni rese possibili dalla memoria virtuale, si può utilizzare la copiatura su scrittura e l'associazione dei file alla memoria.
La copiatura su scrittura prevede che quando un processo viene creato mediante la chiamata di sistema fork (che crea un processo figlio come duplicato

del processo genitore), non vengano duplicate tutte le pagine del processo padre. Il figlio condivide le pagine con il genitore e se tenta di scrivere una pagina condivisa, ne viene creata una copia di sua proprietà.
Con l'associazione dei file alla memoria, si vogliono ottimizzare gli accessi ai file mediante le chiamate di sistema open, read e write. Generalmente ogni accesso al file richiede una chiamata di sistema e un accesso al disco. Con questa tecnica invece una parte dello spazio degli indirizzi virtuali viene associata logicamente a un file facendo corrispondere uno o più blocchi di disco, a una o più pagine di memoria. L'accesso iniziale al file avviene per mezzo del meccanismo di paginazione su richiesta, che determina un'assenza di pagina, cui segue il caricamento di una porzione di file della dimensione di una pagina, in una pagina fisica. In tal modo le operazioni seguenti di lettura e scrittura saranno gestiti come normali accessi a memoria.
Qualora accada che un processo in memoria abbia bisogno del caricamento di una pagina ma non vi siano blocchi di memoria disponibili, bisognerà scegliere una pagina da sostituire, scrivere tale pagina, modificare la tabella delle pagine e scrivere la pagina richiesta sul blocco liberato. Per la sostituzione delle pagine si possono utilizzare diversi algoritmi.
Con l'algoritmo FIFO, la lista delle pagine è strutturata come una coda nel quale il primo elemento a essere inserito, sarà anche il primo a essere estratto, pertanto, la pagina da sostituire sarà sempre la prima della coda. Con l'algoritmo LRU (Last Recently Used), viene sostituita la pagina che non è stata usata per il periodo più lungo di tempo. Tale algoritmo viene realizzato mediante contatori, associando alla cpu un contatore che si incrementa in occasione di ogni riferimento a una pagina o tramite una pila dove ogni volta che si fa riferimento a una pagina, questa viene collocata in cima alla pila stessa. Con l'algoritmo basato su conteggio, si contano il numero di riferimenti che sono stati fatti a ciascuna pagina e si sceglie se sostituire le pagine con conteggio più alto, oppure quelle con il conteggio più basso.
In presenza di più processi occorre stabilire quanti e quali blocchi di memoria associare a ciascuno di essi, tenendo conto del numero massimo di blocchi disponibili e del numero minimo di blocchi da assegnare a un processo. L'assegnazione può essere uniforme, se la memoria libera viene divisa in parti uguali tra i processi, o proporzionale, se la memoria libera si assegna a ciascun processo secondo la sua dimensione.
Quando un processo a bassa priorità non riesce ad ottenere il numero minimo di blocchi di memoria necessari (thrashing), dovrà essere interrotto, in quanto, pur sostituendo una delle sue pagine attive, continueranno ad essere generati dei page fault trap. Tutti questi processi verranno spostati dalla coda dei processi pronti, a quella del dispositivo di paginazione, andando a diminuire l'uso della CPU. Se lo scheduler rileva questo stato di basso uti-

lizzo del processore, andrà ad aumentare il grado di multiprogrammazione, provocando però in realtà un ulteriore incremento del thrashing. Un rimedio consiste nel fornire ai processi i blocchi di memoria di cui necessitano. Per conoscere quanti blocchi servono a un processo si utilizzano diversi metodi come il modello di località o il monitoraggio della frequenza dei page fault. Se la Page Fault Frequency supera un limite superiore precedentemente definito, occorre assegnare a quel processo un blocco di memoria in più, se invece scende al di sotto di un limite inferiore anch'esso precedentemente definito, verrà sottratto un blocco al processo.

Capitolo 5

Il File System

5.1 Interfaccia del File System

Il file system è colui che fornisce il meccanismo per la registrazione e l'accesso a dati e programmi. E' caratterizzato da un insieme di file contenenti dati e da una struttura di directory che organizza i file. Alcuni file system sono dotati anche di partizioni che si occupano di separare grandi gruppi di directory. Il file è la più piccola porzione di memoria logica secondaria. Ogni file è identificato da un nome ed ha degli attributi tra cui: l'identificatore che lo identifica univocamente all'interno del file system, il tipo, la locazione, la dimensione, la protezione, l'ora, la data e l'identificazione dell'utente che l'ha creato, modificato o utilizzato. Ci sono diverse operazioni che si possono eseguire sui file tra cui: la creazione che consiste nel cercare spazio per il file nel file system, la scrittura, la lettura, il riposizionamento dove viene cambiato il valore del puntatore alla posizione corrente affinché punti ad una nuova posizione all'interno del file, la cancellazione e il troncamento che consiste nell'eliminazione del contenuto del file ma non dei suoi attributi. La ricerca del file all'intero di una directory avviene mediante la chiamata di sistema open, che inserisce nella tabella dei file aperti un elemento corrispondente al file aperto. Tale elemento sarà poi rimosso dalla tabella al termine dell'utilizzo del file mediante la chiamata close. Nella tabella dei files aperti, si memorizza il puntatore alla posizione corrente nel file, il contatore delle aperture che viene incrementato ogni volta che si effettua una open su quel file e decrementato per ogni close, la posizione del file sul disco e i diritti d'accesso per fare in modo che ogni processo possa aprire il file secondo diverse modalità.
Per definire il tipo di file, i sistemi Windows dividono ciascun file in due parti separate da un punto: la seconda parte prende il nome di estensione e iden-

tifica il tipo di file. In Unix invece, all'interno dei file viene memorizzato un codice detto magic number che identifica il tipo in modo generico. L'accesso a un file può essere di vari tipi. L'accesso sequenziale consiste nel leggere un'unità dopo l'altra le informazioni contenute nel file. Il puntatore alla posizione corrente viene incrementato di un'unità dopo ogni lettura o scrittura. Altre operazioni consentite sono lo spostamento in avanti o in indietro di un numero fissato di unità, o il riavvolgimento che posiziona il puntatore all'inizio del file. Utilizzano l'accesso sequenziale i nastri magnetici. Un tipo d'accesso alternativo è l'accesso diretto, dove il file è considerato una struttura di elementi logici detti record a dimensione fissa. I programmi accedono direttamente a ciascuno di questi record senza un particolare ordine.
Poiché un file system può essere molto ampio, di solito lo si suddivide in più partizioni. Le directory risiedono nella memoria secondaria e contengono informazioni sui files della partizione. Se tutti i file risiedono nella stessa directory, essa sarà a singolo livello e i files dovranno avere nomi univoci. Nella directory a due livelli invece vi è una directory principale detta Master File Directory che contiene un elemento per ogni utente del sistema. Ciascuno di questi elementi, punta a una directory associata a ciascun utente detta User File Directory, che contiene i file di quel determinato utente. Le directory possono anche avere una struttura ad albero, in tal caso si ha una root directory dalla quale derivano tutte le sottodirectory o una struttura a grafo aciclico, dove si può accedere allo stesso file o alla stessa directory anche da due directory diverse. La condivisione dell'elemento è permesso da un collegamento all'elemento condiviso detto link.
Affinché un filesystem sia accessibile ai processi di un sistema, esso deve essere montato. Nei sistemi Unix, deve pertanto essere specificato un punto di montaggio, che solitamente consiste in una directory vuota alla quale agganciare il file system. Si deve inoltre specificare il dispositivo o la partizione su cui risiede il file system da montare. Il sistema operativo verifica la validità del file system contenuto nel dispositivo e annota nella sua struttura di directory, che un certo file system è stato montato in corrispondenza del punto di montaggio. Nei sistemi Windows, l'utente non deve richiedere il montaggio, in quanto questo è automatico e dopo questa operazione, viene associata una lettera di unità ad ogni partizione accessibile al sistema. Ogni lettera di unità discende da un file system separato. Alcune strutture di directory permettono la condivisione dei file tra più utenti. Si presenta pertanto la necessità di stabilire chi può accedere ai file. L'accesso può essere di vari tipi: lettura, scrittura, esecuzione, aggiunta, cancellazione o elencazione.
La protezione viene utilizzata per impedire l'accesso da parte di utenti non autorizzati, i quali potrebbero cancellare e modificare i file. Nei sistemi monoutente, questa può essere assicurata impostando una password di ac-

cesso all'avvio del sistema. Nei sistemi multiutente, il controllo degli accessi viene effettuato mediante una Access Control List, contenente i permessi associati agli utenti sui file, ovvero, stabilisce se ciascun utente può leggere, scrivere, eseguire, aggiungere, cancellare o elencare un determinato file. Le ACL presentano diversi problemi in quanto possono assumere dimensioni notevoli a seconda del numero degli utenti. La loro costruzione può essere noiosa e il descrittore del file non può essere di dimensione fissa in quanto non si conosce la dimensione che potrà assumere l'ACL associata. Le ACL vengono pertanto divise in 3 classi:

- Proprietario: che individua l'utente che ha creato il file e ne ha il pieno controllo;
- Gruppo: che raggruppa l'insieme di utenti che condividono il file e richiedono accessi di tipo simile;
- Universo: che raggruppa tutti gli altri utenti del sistema.

Nei sistemi Unix, occorrono solo 3 campi per definire la protezione, uno per ogni classe su citata, e ogni campo è definito da 3 bit rxw dove r controlla l'accesso in modalità read, w controlla l'accesso in modalità write e x l'acceso in modalità execute.

5.2 Realizzazione del File System

La realizzazione del file system comporta la scelta dell'interfaccia ovvero dell'aspetto che assumerà agli occhi dell'utente. E' altresì prevista la creazione di algoritmi e strutture dati che permettano di far corrispondere il file system ai dispositivi fisici di memoria secondaria. A tal fine si utilizza una struttura stratificata, dove ogni strato si serve delle funzioni dei livelli inferiori per creare nuove funzioni che verranno sfruttate dai livelli superiori. Il livello più basso è quello del controllo dell'I/O costituito dai driver dei dispositivi il quale si occupa del trasferimento delle informazioni fra memoria secondaria e memoria centrale. Salendo di livello c'è il file system di base che ha il compito di inviare dei generici comandi all'appropriato driver, indicando quali blocchi fisici leggere o scrivere nel disco. Poi troviamo il modulo di organizzazione dei file che traduce gli indirizzi dei blocchi logici in indirizzi dei blocchi fisici e infine il file system logico che gestisce la struttura di directory e i descrittori di file e si occupa della protezione degli stessi. Per la realizzazione del file system si utilizzano diverse strutture dati:

- il blocco di controllo dell'avviamento (boot control block) che contiene informazioni necessarie all'avviamento del sistema operativo da quel disco o partizione;
- i blocchi di controllo delle partizioni che contengono vari dettagli riguardo le partizioni;
- le strutture di directory utilizzate per organizzare i file e i descrittori degli stessi, che contengono molti dettagli riguardo i file.

In memoria sono conservate altre informazioni utilizzate per gestire il file system e migliorare le prestazioni con l'uso della cache. Le principali strutture in memoria sono:

- la tabella delle partizioni che contiene informazioni su ciascuna delle partizioni montate;
- la cache della struttura di directory che contiene informazioni relative a tutte le directory e processi ai quali si è avuto accesso di recente;
- la tabella generale dei file aperti che contiene una copia del descrittore di file per ciascun file aperto;
- la tabella dei file aperti per ciascun processo.

Un disco può essere diviso in più partizioni. Se la partizione non è dotata di file system, è priva di struttura logica pertanto detta raw partition. Le informazioni relative all'avvio del sistema (bootstrap), sono memorizzate in una partizione senza file system, in quanto, all'avvio, non sono stati ancora caricati i driver di dispositivo del file system. Da questa partizione i dati vengono letti in maniera sequenziale. Solitamente, all'avvio, si carica un programma contenuto in una EPROM che avvia il codice contenuto nella partizione o nel blocco di controllo dell'avviamento. Se sul sistema sono utilizzati più sistemi operativi, all'avvio viene caricato un boot loader capace di interpretare diversi file system e di avviare diversi sistemi operativi.
Gli attuali sistemi operativi devono gestire diversi file system contemporaneamente, come per esempio quello del disco e quello di un cd-rom. A tal fine si ricorre ad una soluzione modulare divisa in tre strati: il primo è di alto livello e consiste nell'interfaccia del file system, il secondo è il file system virtuale con il compito di separare le operazioni generiche del file system dalla loro realizzazione definendo un interfaccia Virtual File System uniforme e di gestire anche file system remoti, l'ultimo strato consiste nel file system del dispositivo. Per realizzare una directory, si utilizza una lista lineare che contiene i nomi dei file con puntatori ai blocchi di dati. La ricerca sulla lista è

lineare e per ottimizzare i tempi si utilizza talvolta una cache in cui vengono memorizzate le ultime informazioni utilizzate nelle directory. Altre strutture potrebbero essere la lista ordinata che migliora il tempo di ricerca ma aumenta il tempo per la creazione e cancellazione dei file o ancora i B-Trees.
Ci sono poi 3 modi di assegnazione dello spazio su disco.
L'assegnazione contigua prevede che ogni file debba occupare un'insieme di blocchi contigui sul disco. Per ogni file si memorizza il blocco iniziale e la sua lunghezza in blocchi. Questo tipo di assegnazione causa la frammentazione esterna in quanto assegnando e liberando lo spazio per i file, lo spazio libero dei dischi viene frammentato in tanti piccoli pezzi.
L'assegnazione concatenata prevede che ogni file sia composto da una lista concatenata di blocchi fisici collocati in qualsiasi parte del disco. In questo modo si previene la frammentazione esterna ma lo svantaggio è che tale assegnazione può essere utilizzata in modo efficiente solo con l'accesso sequenziale, infatti, con l'accesso diretto, bisognerebbe partire dall'inizio del file e seguire tutti i puntatori tra blocchi con un notevole calo di prestazioni.
L'assegnazione indicizzata prevede il raggruppamento di tutti i puntatori ai blocchi di un file, in un unico blocco indice. E' adatta anche all'accesso diretto ma le dimensioni del blocco indice potrebbero essere eccessive.
Per gestire lo spazio libero il sistema conserva un elenco dei blocchi non assegnati. Per rappresentare tale elenco si può utilizzare:

- un vettore di bit, che è molto semplice da usare, ma può assumere dimensioni notevoli;
- una lista concatenata, che consiste nel collegare tutti i blocchi liberi, che tuttavia non risulta molto efficiente in quanto per attraversare la lista è necessario leggere ogni blocco;
- il raggruppamento dei blocchi, dove il primo blocco libero conterrà gli indirizzi di n blocchi liberi e di ciascuno di questi ultimi, l'ultimo blocco conterrà a sua volta gli indirizzi di altri n blocchi liberi;
- il conteggio dove per ogni sequenza di blocchi liberi consecutivi, viene memorizzato l'indirizzo del primo blocco e il numero di blocchi liberi.

Per migliorare le prestazioni, i dischi moderni dispongono di una cache all'interno dei controller. Quando viene richiesto un blocco, vengono caricati nella cache anche i blocchi vicini che hanno maggiore probabilità di essere richiesti da un controller. Oltre alla cache sul controller è presente anche una cache delle pagine per i file da mantenere in memoria centrale. Questo permette di impiegare tecniche di memoria virtuale per gestire i dati dei file come pagine

anziché come blocchi di file system. I due algoritmi per la gestione della cache delle pagine sono il free-behind che rimuove una pagina dalla cache non appena viene richiesta la corrispettiva pagina successiva e il read-ahead che a seguito della richiesta di lettura di una pagina, carica in cache anche le corrispondenti pagine successive. L'utilizzo della cache migliora notevolmente le prestazioni, ma le modifiche fatte sulla cache non vengono riportate immediatamente in memoria secondaria. Un crash di sistema potrebbe pertanto portare alla perdita delle modifiche fatte e a uno stato di incoerenza del file system. Per questo motivo al riavvio del sistema viene eseguito il verificatore della coerenza che confronta i dati nelle directory con quelli contenuti nei blocchi dei dischi e corregge eventuali incoerenze.

5.3 Protezione dei File

Disporre di un meccanismo di protezione che controlli e gestisca gli accessi alle risorse, è indispensabile al fine di salvaguardare l'integrità delle stesse, prevenire eventuali violazioni intenzionali dei vincoli d'accesso e migliorare l'affidabilità del sistema.
Un sistema di calcolo è composto da processi e oggetti. Ciascun oggetto è caratterizzato da un nome unico e da delle operazioni. Al fine di proteggere il sistema è opportuno adottare il principio del privilegio minimo che prevede che i processi siano autorizzati ad accedere esclusivamente alle risorse necessarie ad eseguire i propri compiti. A tal fine, si definisce un dominio di protezione che stabilisce i diritti d'accesso su ciascun oggetto, ovvero, determina quali operazioni si possono compiere sugli oggetti. L'associazione tra processo e dominio può essere statica, se la disponibilità delle risorse per un processo è valida per tutta la durata del processo, oppure dinamica nel caso contrario. Un dominio può essere realizzato in diversi modi: ogni utente può essere un dominio e ciò implica che il cambio di dominio avviene al cambio di utente, ogni processo può essere un dominio, in tal caso il cambio di dominio corrisponde all'invio di un messaggio da un processo all'altro e quindi all'attesa di una risposta, ogni procedura può essere un dominio e quindi il cambio di dominio avviene quando si invoca una procedura.
Un modello di protezione corrisponde ad una matrice d'accesso in cui le righe della matrice rappresentano i domini e le colonne gli oggetti. Ciascun elemento della matrice consiste in un insieme di diritti d'accesso. La matrice d'accesso permette di associare i processi ai domini stabilendone i rispettivi permessi. La modifica del contenuto della matrice d'accesso è possibile utilizzando tre operazioni: Copy, Owner e Control. Copy permette di copiare il diritto d'accesso solo all'interno della colonna (cioè per l'oggetto) per la

quale il diritto stesso è definito. Owner permette di aggiungere o rimuovere qualsiasi elemento della colonna. Control permette di modificare gli elementi di una riga.
La matrice d'accesso può essere realizzata secondo 4 schemi:

- Tabella globale: la matrice viene realizzata come una tabella composta da triple del tipo <dominio, oggetto, insieme dei diritti>. Ogni volta che si vuole eseguire una data operazione su un oggetto di un dominio, si controlla se esiste la tripla corrispondente; in caso negativo, si verifica una condizione di eccezione (errore). Lo svantaggio è che la tabella è troppo grande per esser tenuta in memoria centrale ed è pertanto necessario ricorrere a ulteriori richieste di I/O.

- Liste d'accesso per oggetti: per ogni oggetto si crea una lista composta da coppie ordinate <dominio, insieme dei diritti>.

- Liste delle abilitazioni per domini: per ogni dominio si crea una lista di oggetti associata alle rispettive operazioni ammesse sugli oggetti.

- Meccanismo chiave serratura: Ogni oggetto ha una lista di sequenze di bit uniche dette serrature e ogni dominio possiede una lista di sequenze di bit uniche dette chiavi. Un processo in esecuzione in un dominio può accedere a un oggetto solo se chiave e serratura corrispondono.

Può essere talvolta necessario revocare i diritti d'accesso ad oggetti condivisi da diversi utenti. La revoca può essere immediata se ha effetto immediato, ritardata se non inizia immediatamente ma in una data futura, selettiva se viene selezionato un gruppo di utenti ai quali revocare i diritti, generale se vale per tutti gli utenti, parziale se vengono revocati solo alcuni diritti, totale se vengono revocati tutti i diritti, temporanea se ha una durata limitata o permanente se il diritto non può essere riottenuto.

Capitolo 6

I/O e Gestione delle Periferiche

6.1 Virtualizzazione dell'I/O

La varietà dei dispositivi di I/O è sempre crescente tanto da rendere difficile il compito di integrare tali dispositivi nei sistemi operativi. Per risolvere questo problema, si strutturano gli elementi di base dell'architettura di I/O come le porte o il bus in modo uniforme, per potervi connettere una grande varietà di dispositivi. Sfruttando poi i driver dei dispositivi, viene offerta un'interfaccia uniforme per l'accesso agli stessi. I dispositivi comunicano con il calcolatore tramite un punto di connessione detto porta, per mezzo della quale è possibile inviare segnali via cavo o via etere.
Sono 3 gli elementi che caratterizzano l'architettura di un sistema di I/O:

- una porta di I/O costituita da 4 registri: il registro status che indica lo stato della porta, il registro control utilizzato per cambiare alcune modalità di funzionamento del dispositivo (per esempio passando dalla comunicazione half-duplex alla full-duplex), il registro data-in che viene letto dalla cpu per ricevere i dati, e il registro data-out che viene scritto dalla cpu per emettere dati;

- un bus utilizzato per connettere la cpu con la memoria e i dispositivi e un protocollo che specifica che tipo di messaggi può viaggiare sul bus;

- un controllore che consiste in un insieme di componenti elettronici che controllano i segnali presenti nei fili della porta seriale.

L'interazione tra la cpu e un controllore è basato su una negoziazione detta handshaking, che prevede che la cpu legga ripetutamente il registro di stato finché il dispositivo non sia disponibile, segnali nel registro command il tipo di istruzione da eseguire, inserisca dati nel registro data-out ed infine segnali

nel registro command che l'istruzione è pronta. Quando il controllore si accorge di avere un'istruzione da eseguire, segnala poi nel registro di stato che la periferica è occupata. Il controllore legge nel registro command l'istruzione e nel registro data-out i dati da utilizzare. Svolge l'operazione richiesta, azzera il segnale di 'istruzione pronta' nel registro command e indica nel registro di stato che l'operazione è stata eseguita senza errori e che ora la periferica è disponibile. Questa sequenza di operazioni è detta polling.
I controllori sono collegati alla CPU tramite una linea di controllo dell'interruzione. La CPU quando rileva una richiesta su tale linea, salva lo stato dell'elaborazione corrente e esegue una chiamata alla procedura di gestione delle interruzioni, che porta a termine l'elaborazione necessaria ed invia un'istruzione per riportare la CPU nello stato precedente all'interruzione.
Per gestire al meglio le interruzioni è necessario:

- poter differire la gestione delle interruzioni durante l'elaborazioni critiche: a questo scopo, le CPU hanno 2 linee di richiesta delle interruzioni, una per le interruzioni non mascherabili riservata agli errori irrecuperabili, l'altra per le interruzioni mascherabili che può essere disattivata dalla CPU prima dell'esecuzione di una sequenza critica delle operazioni.

- avere un modo efficiente per recapitare un segnale d'interruzione al corretto gestore: a tal fine il vettore delle interruzioni contiene gli indirizzi di memoria degli specifici gestori delle interruzioni. Il meccanismo delle interruzioni accetta un indirizzo, interpretato come uno scostamento relativo a questo vettore.

- poter distinguere tra interruzioni di alta e bassa priorità: a tale scopo si utilizza un sistema di livelli di priorità.

Quando un dispositivo deve trasferire una grande quantità di dati risulta costoso impiegare la CPU, si ricorre pertanto al controllore dell'accesso diretto alla memoria (DMA). Inizia, pertanto, la negoziazione tra DMA e controllore del dispositivo: il controllore invia un segnale sulla linea DMA request quando è pronto per il trasferimento. La cpu scrive in memoria un comando composto da: un puntatore alla locazione dei dati da trasferire, un puntatore alla destinazione dei dati e il numero di byte da trasferire e prima di cedere il bus al DMA, scrive nel controllore DMA l'indirizzo di tale comando. Il DMA prende possesso del bus della memoria e ciò implica che la CPU non possa più accedere alla memoria centrale, pur potendo continuare a elaborare i dati contenuti nella sua cache (fenomeno noto come sottrazione di cicli). Dopo aver preso il controllo del bus di memoria, il DMA manda un segnale sulla linea DMA acknoledgement, il controllore trasferisce i dati e rimuove il segnale dalla linea dma request. Terminato il trasferimento, il controllo del bus

torna al processore. Il DMA può anche utilizzare indirizzi virtuali e compiere trasferimenti di dati tra due dispositivi di I/O associato alla memoria, senza l'intervento della CPU.

6.2 Gestione delle Periferiche

I dispositivi di I/O possono essere divisi in due categorie: a blocchi o a carattere. Un sistema operativo è capace di controllare tutte le periferiche connesse al pc e di gestire eventuali errori fornendo le stesse interfacce tra dispositivi diversi e il resto del sistema (device-independent). Tali interfacce comprendono i comandi accettati dall'hardware, le funzioni che supporta e gli errori che potrebbero essere restituiti. Un'interfaccia uniforme viene offerta sfruttando i driver dei dispositivi i quali comunicano con il calcolatore tramite un punto di connessione detto porta, per mezzo della quale è possibile inviare segnali via cavo o via etere. Un esempio pratico è il File System, che lavora con le periferiche e non deve avere differenti modalità di accesso alle periferiche, nonostante queste possano essere sia a blocchi che a carattere. Il File System lavorerà con dispositivi a blocchi astratti, lasciando la vera e propria implementazione ai device driver.
Le unità di I/O sono generalmente composte da un componente elettronico detto device controller che permette al dispositivo di interfacciarsi con il PC e un componente meccanico che è il dispositivo stesso. Il sistema operativo non tratta mai direttamente con il dispositivo ma sempre con il controller e gestisce l'I/O attraverso una strutturazione a livelli. I livelli più bassi servono a mascherare le specifiche hardware dei dispositivi, mentre quelli superiori servono per fornire un interfaccia comune agli stessi. L'obiettivo è creare una device independence al fine di permettere ai programmi di eseguire operazioni su qualsiasi device (per esempio un harddisk o un floppy) senza modificare il codice del programma. Per eseguire operazioni in modo indipendente sul dispositivo, deve esserci uniform naming, deve pertanto essere creata un'indipendenza logica dal dispositivo sul quale si sta lavorando. Inoltre, per mezzo dell'error handling, un dispositivo può individuare gli errori e correggerli al livello più basso e, solo se il problema non può essere risolto in questo modo, i livelli più alti ne vengono informati e l'errore verrà gestito in modo indipendente dal dispositivo. Il sistema operativo deve inoltre gestire la virtualizzazione delle periferiche per permettere a più utenti di utilizzare i dispositivi di I/O e deve occuparsi di gestire i possibili conflitti tra utenti. Può essere strutturato nei seguenti livelli: interrupt handler, device driver, device indipendence OS software, user level software. Al livello più basso interrupt handler, vengono gestiti gli interrupt quando deve essere

segnalato al sistema operativo il verificarsi di un determinato evento. Fanno parte del livello device driver, i controller caratterizzati dai registri usati per trasferire dati e comandi e il disk driver che è colui che conosce i comandi da utilizzare per comunicare con il dispositivo. Il compito di un driver è di accettare richieste astratte provenienti dal device-independent software e controllare che queste siano eseguite correttamente. Quando i comandi da utilizzare sono stati scelti, vengono copiati nei registri del dispositivo. Il driver attende che l'operazione restituisca il risultato, controlla che non ci siano stati errori e rende il risultato al device independet layer. A livello device indipendence OS software, le funzioni base permettono al sistema operativo di eseguire operazioni di I/O comuni a tutti i dispositivi e di fornire un'interfaccia comune all'user-level software. Le funzioni che questo livello mette a disposizione sono: il naming e protezione, il device-independent block-size, il buffering, l'allocazione e il rilascio dei dispositivi, l'error handling. Il naming gestisce il mapping tra i nomi simbolici dei driver (es. LPT1, A:) e i driver a cui si riferiscono, la protezione previene gli accessi indesiderati ai dispositivi (errori, intrusioni), il device-independent block-size maschera la differenza che possono avere differenti sector size e la differenza di dimensione dei dispositivi che possono essere a blocchi o a carattere, il buffering si occupa di gestire letture successive di grandi blocchi oppure fornire dati tutti insieme, l'allocazione e il rilascio permette ai dispositivi di poter essere usati da uno o più utenti, l'error handling gestisce gli errori al livello più basso. A livello user level software i sistemi operativi mettono a disposizione dei programmatori delle librerie di funzioni che permettono, tra l'altro, di usare i dispositivi connessi al PC.